Combining Advantages from Parameters in Modeling and Control of Discrete Event Systems

Luiz F. P. Southier[1], Muriel Mazzetto[1], Dalcimar Casanova[1], Marco A. C. Barbosa[1], Luis S. Barbosa[2] and Marcelo Teixeira[1]

***Abstract*—Although *Finite-State Automata* (FSA) have been successfully used in modeling and control of *Discrete Event Systems* (DESs), they are limited to represent complex and advanced features of DESs, such as context recognition and switching. The literature has suggested that a FSA can nevertheless be enriched with parameters properly collected from the modeled system, so that this favors design and control. A parameter can be embedded either on transitions or states. However, each approach is structured within a specific framework, so that their comparison and integration are not straightforward and they may lead to different control solutions, modeled, computed and implemented using distinct strategies. In this paper, we show how to combine advantages from parameters in modeling and control of DESs. Each approach is structured and their advantages are identified and exemplified. Then, we propose a conversion method that allows to translate a design-friendly model into a synthesis-efficient structure. Examples illustrate the approach.**

***Index Terms*—Discrete Event Systems, Formal Modeling, Modeling Integration, Event-parameter, State-parameter, Control.**

I. Introduction

Modern automation systems aim to transform material into products by properly, flexibly and efficiently integrating people, equipments and technology [1]–[3]. It is expected that factory components can interact with each other and with the environment in a concurrent manner, sharing resources and behaving in a safe, controllable and maximally permissive way. In conjunction, those features make it hard the task of programming industrial controllers, as traditional paradigms for software development are usually inappropriate. An alternative is to adopt *model-driven* strategies to express system behavior and requirements [4]. In this case, automated operations can be processed in order to calculate a controller that holds properties of interest.

When an industrial process is seen as a *Discrete Event System* (DES) [5], the control objective is to obtain sequences of events to be allowed under control. Events are assumed to occur spontaneously in the system *plant*, and they are restricted by *specifications*, which are in general modeled using *Finite State Automata*. Then, formal approaches, such as the *Supervisory Control Theory* (SCT) [6], can be applied to synthesize controllers to be finally implemented in hardware.

Despite their practical relevance and formal background, FSA face significant limitations when modeling large and complex systems. Advanced features of flexible DES, such as context recognition and switching, are difficult to be expressed through ordinary FSA and they are usually associated with complex models, both in terms of modeling and processing [2], [7], [8]. *Parameterized* FSA allow to address complexity issues in modeling and control of DESs. A so-called *parameter* is an engineered argument, embedded on a modeling formalism, that captures and carries context semantics throughout a FSA. This mechanism expands the information potential of a DES model and extends its control techniques to cover a broader class of problems [8]–[10].

Technically, a parameter can be embedded either on transitions [10] or states [9] of a DES model. *State-parameterized FSA* (SpFSA) can be modeled by using *Extended Finite State Automata* [11], structures that implement control by disabling events based on the evaluation of logical formulas that manipulate variables. Differently, *Event-parameterized FSA* (EpFSA) are mechanisms that systematically map events into sets of new events, called *parameterized events*, which carry a given context semantic for the modeled system.

In theory, both SpFSA and EpFSA play a similar role in modeling and control of DES, so that their choice should be straightforward. However, it does not exist so far in the literature an explicit way to compare them and evidence their advantages. As each approach is structured within a specific framework, their integration is not direct and they may lead to different control solutions, modeled, computed and implemented using distinct strategies [9], [10], [12].

It has been reported that EpFSA benefit modeling and synthesis [10], besides to be modular [13] and to reduce implementation costs [12]. But, in this case, the entire parametrization structure depends on an engineer to be constructed. Differently, SpFSA are more suitable for modeling as they allow to express control conditions by simple formulas [8], [9], [11]. However, the variable structure of a SpFSA is not natively modular, which may complexify synthesis and implementation.

This paper shows how to combine advantages from SpFSA and EpFSA. A conversion process is formally structured and algorithms are provided to systematically extract meanings (parameters) from the states of a SpFSA, transferring them to an event-based structure that leads to an equivalent EpFSA. Then, we show how the resulting EpFSA can be used as input to efficient synthesis frameworks [13], [14] and related implementation options [12]. The proposed conversion method is illustrated by an example of a manufacturing system with

Luiz F. P. Southier, Muriel Mazzetto, Dalcimar Casanova, Marco A. C. Barbosa and Marcelo Teixeira are with the Universidade Tecnológica Federal do Paraná, Pato Branco, Brazil ({luizsouthier, murielmazzetto}@alunos.utfpr.edu.br,{dalcimar,mbarbosa, marceloteixeira}@utfpr.edu.br); Luís S. Barbosa is with the Universidade do Minho, Minho, Portugal (lsb@di.uminho.pt).

978-1-7281-0303-7/19/$31.00 ©2019 IEEE

intermediate buffering of materials.

The manuscript is structured as follows: the mathematical background is presented in Section II; the main results are introduced, assessed and exemplified in Section III; and Section IV brings some conclusions and perspectives.

II. Background

Many real systems share the feature of being event-driven, i.e., their evolution in time is guided by the occurrence of asynchronous signals, called *events*, in opposition to time-driven behaviors. Systems that share these features are called *Discrete Event Systems* (DES) [5] and they cover a wide range of domains, such as robotics, manufacturing, logistics, etc.

A DES can be modeled using *formal languages*. *Events* define the basic structures of languages, and they are elements of a finite set, Σ, called the *alphabet*, such that Σ^* denotes the set of all *strings* possibly built using events in Σ, including the *empty string* ε. Then, any subset $L \subseteq \Sigma^*$ is called a *language*, and it is said to be regular if it can be represented by the well-known *Finite-State Automata* (FSA) [5].

A FSA is a 5-tuple $A = (\Sigma, Q, q^\circ, Q^\omega, \gamma)$, where Σ is the alphabet; Q is the set of states; $q^\circ \in Q$ is the initial state; $Q^\omega \subseteq Q$ is the subset of marked states; and $\gamma : Q \times \Sigma \to Q$ is a partial transition function. Notation $q_1 \xrightarrow{\sigma} q_2$ means a transition from $q_1 \in Q$ to $q_2 \in Q$ with the event $\sigma \in \Sigma$.

Two FSA can be combined by the usual *synchronous composition*, denoted by $||$, which synchronizes shared events and interleaves the others [5]. In the following, notation $A^{\|}$ means the composition of a set $A = \{A_1, ...A_n\}$ of FSA, i.e., $A^{\|} = A_1 \| ... \| A_n$. The language recognized by A is denoted $\mathcal{L}(A)$, and $\mathcal{L}^\omega(A) \subseteq \mathcal{L}(A)$ is the marked language, which in this paper is associated with the idea of tasks that are completed by the system modeled by A.

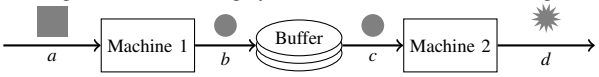
When designing DES, the system model can be represented by a composition of FSA, denoted by $G = \|_{j=1}^{m} G^j$, which is called the *plant model*. The plant is expected to be restricted by a specification E, so that it can behave as intended under control. E can also be designed by a set $E = \|_{i=1}^{n} E^i$ of FSA and composed to G afterwards. In this way, the composition $K = G \,||\, E$, such that $\mathcal{L}^\omega(K) \subseteq \mathcal{L}^\omega(G)$, materializes the control actions on G exactly as projected by the engineer, so it can be implemented or used as input to synthesis frameworks, such as *Supervisory Control Theory* [6].

In this paper, we are interested in the steps that precede synthesis, i.e., we exploit different methods to obtain K. Although further verification, synthesis and implementation are not considered here, the way K is obtained guides the options for the next steps, and this is discussed properly along the paper, whenever convenient.

A. Example of a simple manufacturing system

For illustration, consider a DES composed by two machines, 1 and 2, interconnected by an intermediate buffer as in Fig. 1. Machine 1 receives external raw material (event a), produces workpieces and puts them on a buffer (event b). Then, machine 2 takes workpieces from the buffer (c), manufactures, and removes them from the system (d).

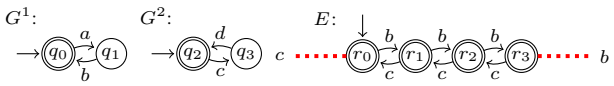
Fig. 1. Manufacturing system with intermediate buffering



Machines 1 and 2 can be respectively modeled by the 2-state FSA G^1 and G^2 shown in Fig. 2, so that the system plant can be modeled by the composition $G = G^1 \,||\, G^2$.

For control, it is assumed that the buffer has capacity of 3 workpieces, and one aims to control overflow and underflow in the buffer. Specification E in Fig. 2 is modeled to prevent overflow and underflow. Dashed lines are used to illustrate the events disabled by E.

Fig. 2. Plant and specification models for the example



Note that the event b is disabled when the buffer is full (state r_3) and event c when the buffer is empty (state r_0). Then, the composition $K = E \,||\, G$, with 16 states, expresses the behavior expected under control.

B. A motivating gap in using ordinary FSA models

Despite the recognized role played by FSA for automation systems modeling and control, they face significant limitations when applied on real industry-scale problems. It can be shown [8]–[10], [13]–[15] that workflows commonly found in factory floors, such as recycling, buffering, parallel manufacturing, etc., may require hundreds of thousands of states to be properly expressed by ordinary FSA.

As this is a manual nonautomated task, memorizing complex sequences of events and states relies entirely on the designer and it is not rarely unworkable. Alternatively, the literature [9], [10] has suggested the possibility of providing extra *information* or *meaning* (in this paper called *parameter*) about particular parts of a DES. The use of parameters allows to identify and isolate certain *contexts* from others, along the system model. When properly engineered, this parameterization can simplify modeling. Two formalisms that allow embedding parameters in FSA are presented in the following.

C. Parameterization of States

State-parameterized FSA (SpFSA) are similar to ordinary FSA, but their transitions include *formulas* over *variables*. Formally, a SpFSA can be expressed as a tuple $A_{\mathfrak{E}} = (\Sigma, V, Q, Q^\circ, Q^\omega, P_V, \gamma)$ [9], [11] where Σ is the alphabet of events; $V = \{v_1, ..., v_n\}$ is the set of variables; Q is the set of states; $Q^\circ \in Q$ is the set of initial states; $Q^\omega \subseteq Q$ is the subset of marked states; P_V is the set of formulas over V; and $\gamma : Q \times \Sigma \times P_V \to Q$ is the transition relation that leads from a state to another in Q, with an event taken from Σ and formula taken from the set of formulas P_V.

In this definition, a variable v is an entity with a finite domain $Dom(v)$ and an initial value $v^o \in Dom(v)$. Then, $V = \{v_1, \ldots, v_n\}$ has a domain $Dom(V) = Dom(v_1) \times \cdots \times Dom(v_n)$. In order to manipulate variables, SpFSA define a set of formulas $P_V = \{p_1, \ldots, p_m\}$. In conjunction, variables and formulas establish a new transition mechanism, also denoted differently: a transition from $q_0 \in Q$ to $q_1 \in Q$, with $\sigma \in \Sigma$ and $p \in P_V$ is exposed as $q_0 \xrightarrow{\sigma:p} q_1$. In order to differentiate variable values before and after a transition, a *next-state* variable set $V' = \{v'_1, \ldots, v'_n\}$, with $Dom(V) = Dom(V')$, is associated to V. Then, $\bar{v} \in Dom(v)$ and $\bar{v}' \in Dom(v')$ mean respectively the value assumed by v in the current and next state. For variables in conjunction, the samples $\hat{v} = (\overline{v_0}, \ldots, \overline{v_n}) \in Dom(V)$ and $\hat{v}' = (\overline{v_0}', \ldots, \overline{v_n}') \in Dom(V')$ are called *valuations*.

A formula $p \in P_V$ can be used either to *update* or *test* variable values upon transitions. Updates allow to handle context switching in the plant, while tests are more related with control restrictions. An update p aims to replace the valuation $\hat{v}$ associated to the current state, to a new valuation $\hat{v}'$ in the reached state, which is denoted by $\hat{v}' = p(\hat{v})$. A valuation $\hat{v}$ is said to be *valid* for p if $\hat{v} \in Dom(V)$, and $p(\hat{v}) \in Dom(V')$.

Differently, test formulas (or *guards*) do not change any variable value, they simply test values associated to the current state, disabling or not the transition depending on the test result, i.e., $p(\hat{v}) = true$ or $p(\hat{v}) = false$. Therefore, any valuation $\hat{v} \in Dom(V)$ is valid over test formulas. When a transition does not implement any formula, or it implements only tests, then $\hat{v}' = \hat{v}$ is implicit, and it is hidden in this paper for the sake of clarity. In this case, the transition is said to be *passive*. Otherwise, it is said to be *active*.

To exemplify, let v be a variable with $Dom(v) = \{1, 2, 3\}$, $V = \{v\}$, and let $v' = v + 1$ be a formula $p_1 \in P_V$ that changes the current value of v by adding 1. If $\hat{v} = 1$, then $p_1(\hat{v}) = 2$ and $\hat{v}$ is valid with respect to p_1. If $\hat{v} = 3$, then $p_1(\hat{v}) = 4 \notin Dom(V')$, and $\hat{v} = 3$ is not valid for p_1. Now, let $p_2 \in P_V$ be a test formula $v > 2$. Then, $p_2(\hat{v})$ is *true* for the valuation $\hat{v} = 3$, and *false*, otherwise.

In this paper, plants implement only updates, while specifications only test values. It is furthermore assumed that updates are all *exact*, i.e., for each valuation $\hat{v}$, an update $p(\hat{v})$ leads to a unique valuation $\hat{v}'$. This differs from the literature that handles abstractions, for example, which has to address nondeterminism of variable values [9]. Also, we consider only *convergent* updates, i.e., two updates cannot implement divergent changes on a same variable. Here, however, plants are not required to include only valid valuations, as invalid values are removed by our conversion algorithms.

Remark that, so far, a SpFSA is exposed in its *implicit* form, i.e., including all its formulation structure. In this case, the valuations to be associated with each state is unknown until each formula is in fact evaluated. After that, the SpFSA becomes *explicit*, as each variable value is revealed for each state and the state-space is unfolded.

In its explicit form, a SpFSA $A_{\mathfrak{E}}$ is seen as an ordinary FSA $A^{\bullet}_{\mathfrak{E}} = (\Sigma, Q_{\bullet}, Q^{\circ}_{\bullet}, Q^{\omega}_{\bullet}, \gamma_{\bullet})$, such that $Q_{\bullet} = Q \times Dom(V)$; $Q^{\circ}_{\bullet} = Q^{\circ} \times \{(v^o_1, \ldots, v^o_n)\}$; $Q^{\omega}_{\bullet} = Q^{\omega} \times Dom(V)$; and $\gamma_{\bullet} : Q_{\bullet} \times \Sigma \to Q_{\bullet}$ assumes the form of $(q_0, \hat{v}) \xrightarrow{\sigma} (q_1, \hat{v}')$ if there is $q_0 \xrightarrow{\sigma:p} q_1$, with $p(\hat{v}) = true$ and $\hat{v}' \in Dom(V')$.

Thus, a DES plant can be modeled by an implicit SpFSA $G_{\mathfrak{E}}$, enriched with context semantics updates, and its explicit version, $G_{\mathfrak{E}}\bullet$, is assumed to be such that

$$L(G_{\mathfrak{E}}\bullet) = L(G). \tag{1}$$

If furthermore $E_{\mathfrak{E}}$ expresses the same control rules as E, then $\mathcal{L}(K_{\mathfrak{E}}{}^{\bullet}) = \mathcal{L}(K)$, for $K = G \| E$ and $K_{\mathfrak{E}} = G_{\mathfrak{E}} \| E_{\mathfrak{E}}$.

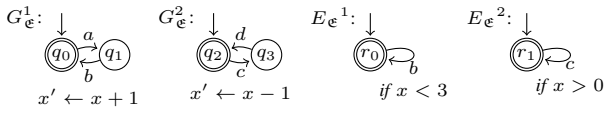
In terms of control, both implicit and explicit SpFSA can be used in synthesis, and both versions lead to the same control solution. The implicit SpFSA controller can be calculated as in [9], while its explicit form can be obtained through the conventional method as in [6].

1) Example with SpFSA*:* Here, we show how the example presented in Section II-A can be reintroduced by using SpFSA and we highlight possible benefits from this reconstruction.

Let the FSA G^1 and G^2 from Figure 2 be now modeled respectively by $G^1_{\mathfrak{E}}$ and $G^2_{\mathfrak{E}}$ in Fig. 3. Structurally, they are essentially the same, except that $G^1_{\mathfrak{E}}$ and $G^2_{\mathfrak{E}}$ update a variable $x \in V$, with domain $Dom(x) = \{\text{-}1, 0, 1, 2, 3, 4\}$ and initial value $x^o = 0$, which has been created to memorize the number of workpieces in the buffer, and it is updated by formulas on transitions with the events b (insertion) and c (removal).

The benefits brought by updating x in the system plant can really be seen when remodeling E. Overflow and underflow can now be prevented respectively by the modular specifications ${E_{\mathfrak{E}}}^1$ and ${E_{\mathfrak{E}}}^2$ depicted in Fig. 3.

Fig. 3. SpFSA models for the example



Note that, now, the $n + 1$-state model E can be simply designed by single self-looped states r_0 and r_1 that test the values of x before enabling the events b and c, so that overflow and underflow can be equivalently avoided.

Then, the plant is now modeled by the composition $G_{\mathfrak{E}} = G^1_{\mathfrak{E}} || G^2_{\mathfrak{E}}$, and the specification is modeled by $E_{\mathfrak{E}} = {E_{\mathfrak{E}}}^1 || {E_{\mathfrak{E}}}^2$, so that $K_{\mathfrak{E}} = G_{\mathfrak{E}} \,||\, E_{\mathfrak{E}}$ expresses the expected behavior under control, with 16 unfolded states, therefore the same number as K. It can be checked (by language inclusion, for example) that $\mathcal{L}(K_{\mathfrak{E}}{}^{\bullet}) = \mathcal{L}(K)$, which means that the computational cost to process both models is the same, but the modeling of $K_{\mathfrak{E}}$ is much simpler and it remains the same, for any buffer size to be considered.

2) A motivating gap in using SpFSA *models:* Despite possible modeling advantages brought by the easy way conditions and behaviors are expressed, SpFSA are entities that do not fully exploit modularization. There are variable abstraction techniques [9] that work modularly [7] by removing unnecessary variables from control synthesis, chosen according to the

role they play in control. This reduces a lot the computational effort needed to process the SpFSA-based synthesis.

However, they do not work with partial abstractions, i.e., abstractions that, besides removing unnecessary variables totally, also remove unnecessary parts of the domain of a necessary variable. Partial abstractions are more difficult to be constructed, as a domain is intrinsically inseparable, so that its is usually taken entirely for synthesis. When the variable domain is large, its transformation to the explicit form, in combination with other variables, leads to huge states-spaces, necessary for algorithmic treatment. In some extent, this prevents modeling advantages to be propagated for synthesis and implementation. Parallel advantages can be taken by converting SpFSA into *Event-parameterized FSA*, which are presented in the following.

D. Parameterization of Events

Event-parameterized FSA (EpFSA) are state-machines that use a different mechanism to store parameters of a DES model. Their premises and purposes are similar to SpFSA but, instead of using variables to store context semantics, they embed it on a refined alphabet of events, systematically mapped from the original set of events. The result is a model that implicitly stores the same information as a SpFSA, however using a different construction mechanism, hopefully more modular.

Formally, a EpFSA maps each event $\sigma \in \Sigma$ into a set of *parameterized events* $\Delta^\sigma = \{\delta_1, \delta_2, \ldots, \delta_n\}$, and each δ aims to store certain context semantics, which is to be further defined. In this way, Σ becomes a *reference event set* for a dilated, *parameterized event set* $\Delta = \bigcup_{\sigma\in\Sigma} \Delta^\sigma$.

The mapping from Δ to the reference Σ can be implemented by $\Pi : \Delta^* \to \Sigma^*$, defined recursively such that $\Pi(\epsilon) = \epsilon$ and $\Pi(t\delta) = \Pi(t)\sigma$ for $t \in \Delta^*$, $\delta \in \Delta^\sigma$ and $\sigma \in \Sigma$ [10]. This map can be generalized to any language $\mathcal{L}_\mathfrak{D} \subseteq \Delta^*$ by $\Pi(\mathcal{L}_\mathfrak{D}) = \{s \in \Sigma^* | \exists t \in \mathcal{L}_\mathfrak{D}, \Pi(t) = s\}$.

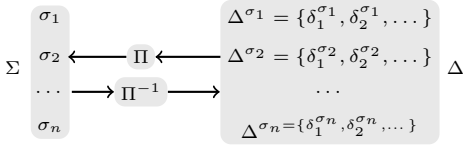
Inversely, $\Pi^{-1} : \Sigma^* \to 2^{\Delta^*}$ implements the mapping from the reference set Σ to the dilated domain Δ. It can be defined as $\Pi^{-1}(s) = \{t \in \Delta^* | \Pi(t) = s\}$, and extended to any language $\mathcal{L}$ by $\Pi^{-1}(\mathcal{L}) = \{t \in \Delta^* | \Pi(t) \in \mathcal{L}\}$.

The extension of this mapping to FSA, instead of languages, follows the same idea, i.e., given a FSA A, the map $\Pi^{-1}(A)$ replaces the event of every transition in a FSA A by the respective set of parameterized events in $A_\mathfrak{D}$. Inversely, $\Pi(A_\mathfrak{D})$ recovers each original event, so that

$$\Pi(\Pi^{-1}(A)) = A \tag{2}$$

follows by construction [10]. The process of events dilatation and recovering is depicted in Fig. 4.

Fig. 4. Event dilatation and recovering



From now, we assume that a DES plant G is modeled by a EpFSA $G_\mathfrak{D}$ that expresses G with enriched context semantics carried by parameterized events, i.e., $G_\mathfrak{D} = \Pi^{-1}(G)$, such that $\Pi(\Pi^{-1}(G)) = G$ follows from (2). Note that, with a dilated alphabet, a transition may enable more than one instance for each event in the reference set. This means that the plant model may recognize different context semantics, but it is unable to choose which one applies at every step.

The precise choice among the instances $\{\delta_1, \delta_2, \ldots, \delta_n\}$ of an event $\sigma \in \Sigma$ depends on constructing an additional model to *filter* $G_\mathfrak{D}$. A filter is denoted in this paper by $H_\mathfrak{D}$ and it has the unique role of imposing *context switching* to the plant. That is, a filter has no intention to disable events completely in the plant, as a specification does. Instead, it simply chooses which parameterized event $\delta_i \in \Delta^\sigma$ should occur when they are dubious. Here, the filter is assumed to be *precise*, i.e., for each set Δ^σ eligible upon a transition, $H_\mathfrak{D}$ chooses one, and only one of them to remain eligible. It represents the context to be enabled and all others are disabled.

Thus, let the plant G be expressed by $G_\mathfrak{D} \| H_\mathfrak{D}$, such that,

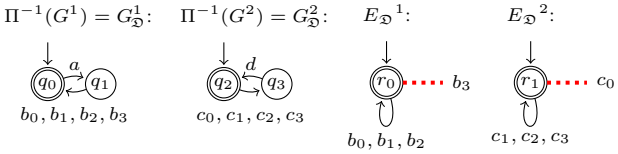
$$\Pi(G_\mathfrak{D} \| H_\mathfrak{D}) = G. \tag{3}$$

Let also $E_\mathfrak{D}$ be a specification expressing the same control rule as E (and indirectly $E_\mathfrak{E}$). Then, it is expected that $\mathcal{L}(\Pi(K_\mathfrak{D})) = \mathcal{L}(K)$, for $K = G \| E$ and $K_\mathfrak{D} = G_\mathfrak{D} \| H_\mathfrak{D} \| E_\mathfrak{D}$. Under this equality and under the assumption that $H_\mathfrak{D}$ is precise, it can be shown [10] that either $K_\mathfrak{D}$ or K can be used as input to any conventional synthesis framework [6] and the resulting control solution is equivalent.

In comparison with FSA, EpFSA are more efficient to design and memorize contexts, at the price of modeling the filter $H_\mathfrak{D}$. However, $H_\mathfrak{D}$ is expected to be modular, as it defines a non-homonym set of events that can be modularly distinguished from each other [10], so that $H_\mathfrak{D}$ can be modeled by $H_\mathfrak{D} = H_{\mathfrak{D}1} \| \ldots \| H_{\mathfrak{D}m}$. Furthermore, EpFSA enable for modular control that can be combined with abstraction-based synthesis [13] and disjoint implementation [12].

1) Example with EpFSA*:* For the example in Section II-A, the corresponding EpFSA models are depicted in Fig. 5.

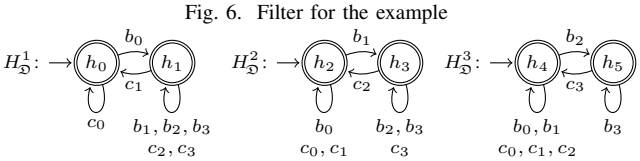
Fig. 5. EpFSA models for example



Events b and c are parameterized such that $\Delta^b = \{b_0, b_1, b_2, b_3\}$ and $\Delta^c = \{c_0, c_1, c_2, c_3\}$, for them to be able to carry extra information about the number of workpieces present in the buffer. Observe, nevertheless, that $G^1_\mathfrak{D}$ and $G^2_\mathfrak{D}$ do not recognize how many workpieces in fact the buffer has, as they enable any parametrized instance of b and c. For example, transition $q_1 \xrightarrow{b_0,b_1,b_2,b_3} q_0$ is labeled with all parameters for b, while they actually should be properly ordered.

This ordering depends on $H_{\mathfrak{D}}$ to choose which parameter should uniquely occur. For the plant in Fig. 5 the filter $H_{\mathfrak{D}}$ can be designed by the set of automata shown in Fig. 6.

Fig. 6. Filter for the example

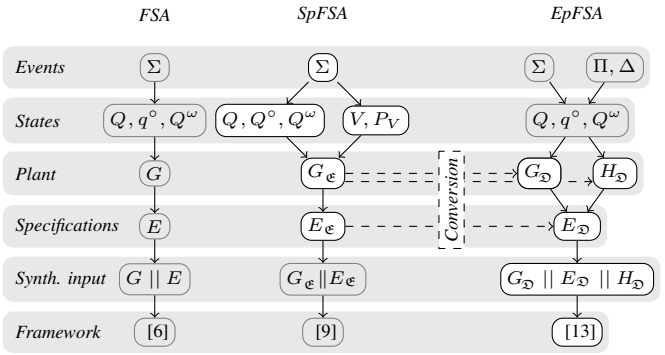


When composed with $G^1_{\mathfrak{D}}$ and $G^2_{\mathfrak{D}}$, $H_{\mathfrak{D}}$ leads to a particular case of parameterized plant that keeps a single string $t \in \mathcal{L}(G^1_{\mathfrak{D}} \,||\, G^2_{\mathfrak{D}} \,||\, H_{\mathfrak{D}})$ for each corresponding $s \in \Sigma^*$, while still simplifying the specification $E_{\mathfrak{D}}$ with respect to E.

III. Proposed Conversion Method

In this section, we show how advantages from SpFSA and EpFSA can be combined. The idea is to conduct modeling using SpFSA, which are then converted into EpFSA for posterior modular synthesis and implementation. To illustrate better where the proposed conversion method fits, Fig. 7 structurally compares FSA, SpFSA and EpFSA.

Fig. 7. Structural comparison of FSA, SpFSA and EpFSA



The first column illustrates the conventional case where a DES and its specifications are respectively modeled by the FSA G and E, where $K = G \,||\, E$ is the synthesis input using the classic monolithic SCT framework [6].

The second column applies SpFSA to address modeling. It uses the same event set Σ as for FSA, but it is helped in modeling by variables and updates. The result is that FSA G and E are now expressed by SpFSA $G_{\mathfrak{E}}$ and $E_{\mathfrak{E}}$, which leads to a composition $K_{\mathfrak{E}}$ that can be used as synthesis input for the algorithm in [9].

The third column structures modeling on a different, dilated, set of evens, Δ. This leads to a plant $G_{\mathfrak{D}}$ that is complemented with $H_{\mathfrak{D}}$ for context recognition. In the same way, E turns to be modeled by $E_{\mathfrak{D}}$ and the synthesis input is then given by the composition $K_{\mathfrak{D}} = G_{\mathfrak{D}} \,||\, H_{\mathfrak{D}} \,||\, E_{\mathfrak{D}}$, such that $\Pi(K_{\mathfrak{D}})$ is expected to be equivalent to K [10].

Summarizing, it is expected that FSA, SpFSA and EpFSA lead to equivalent synthesis inputs. However, SpFSA are more suitable for modeling, while EpFSA are advantageous have the advantage of keeping modularity by renaming events according to contexts, which in synthesis is essential to split computation in smaller, simpler, parts [13].

Next, we show a conversion method that allows to conduct modeling tasks using SpFSA, and convert them into corresponding EpFSA for further steps of control engineering. Figure 7 highlights in white the conversion flow.

A. Proposed Conversion Method

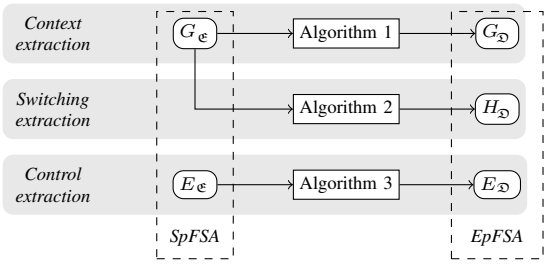
Initially, let us recall that, in a SpFSA, a transition with a given event $\sigma \in \Sigma$ may implement formulas $p \in P_V$ that change or test variables values $\{\overline{v_0}, \ldots, \overline{v_n}\} = \hat{v} \in Dom(V)$, leading to a deterministic valuation $\{\overline{v_0}', \ldots, \overline{v_n}'\} = \hat{v}' \in Dom(V')$ at every state.

The valuations $\hat{v}$ and $\hat{v}'$ are parameters that represent the DES context in current and next states, respectively. In order to reproduce the same effect using parameterized events, we have to concatenate each original event with parameters describing current and reachable contexts. This notion leads to the new set of parameterized events.

Systematically, if there is no context switching upon a transition, i.e., $\hat{v} = \hat{v}'$, then there is no need for creating a parameterized event. However, if $\hat{v} \neq \hat{v}'$, then at least one variable value will change upon the transition and a parameter representing it must be constructed in the resulting EpFSA.

This notion is represented in this paper by the combination $\sigma_{v\bar{v}}$, that is: when a transition with σ is capable of changing the value of v from $\bar{v}$ to $\bar{v}'$, and $\bar{v} \neq \bar{v}'$, then the current context is carried by the event $\sigma_{v\bar{v}}$. By doing this for all states and all possible variable changes, an EpFSA can preserve exactly the same semantic of updates a SpFSA, but using a distinct mechanism, exploiting transitions instead of states.

Fig. 8. Conversion process from SpFSA to EpFSA



Thus, the conversion method proposed in this paper can be conducted by the three-steps procedure shown in Fig. 8. The first step extracts all possible contexts that can be updated by the SpFSA plant $G_{\mathfrak{E}}$ (Algorithm 1). This is more related to the coverage of the variable domain and it leads to a EpFSA that may not be precise. For that, it requires a filter, which is constructed in the second step by extracting the context switching behavior from variable updates (Algorithm 2). Finally, the third step maps the control rules imposed by the specification models, from SpFSA to a EpFSA (Algorithm 3). The conversion process is introduced, discussed and exemplified in the following.

1) Context Extraction: for context extraction, Algorithm 1 initially constructs a structure of states identical to the input SpFSA. As the new alphabet and transition relation are undefined at this point, they are started as empty.

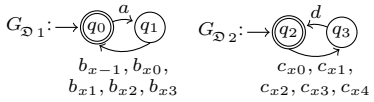
Then, all active transitions are read from the input SpFSA $G^i_{\mathfrak{E}}$ (line 6) in order to identify and construct the sets of parameterized events and corresponding transition structure for the output EpFSA $G^i_{\mathfrak{D}}$. For each valuation $\hat{v} \in Dom(V)$ that is valid with respect to the update formulas in the SpFSA transition, the algorithm compares current and next values of each variable $v \in V$, i.e., $\bar{v} \in \hat{v}$ and $\bar{v}' \in \hat{v}'$. If the values are different ($\bar{v} \neq \bar{v}'$), then the context has switched and a new parameterized event $\sigma_{v\bar{v}}$ is created to report this switching to the new alphabet Δ^σ (line 9). Then, a new transition labeled $\sigma_{v\bar{v}}$ is added to $G^i_{\mathfrak{D}}$ (line 10).

Afterwards, all passive transitions are read to construct $G^i_{\mathfrak{D}}$. In case an event σ has not yet been parameterized, i.e., $\Delta^\sigma = \emptyset$, the transition is added without changes to $G^i_{\mathfrak{D}}$ (line 17) and σ is added to Δ^σ. Otherwise, i.e., $\Delta^\sigma \neq \emptyset$, σ is replaced by all its parameterized instances Δ^σ and the transition is added to $G^i_{\mathfrak{D}}$ (line 21). At the end (line 26), all sets of parameterized events are added to $\Delta_{\mathfrak{D}}$.

As a result, Algorithm 1 identifies all possible contexts in $G^i_{\mathfrak{E}}$ and reproduces the same effect in $G^i_{\mathfrak{D}}$, using a different mechanism that is free from formulas and variables. In terms of modeling benefits and decision making in control, they both are expect to contribute in very similar ways, but their constructions are different, which may return further advantages.

a) Example: For the input models $G^1_{\mathfrak{E}}$ and $G^2_{\mathfrak{E}}$ in Fig. 3, it is possible to obtain the corresponding EpFSA $G^1_{\mathfrak{D}}$ and $G^2_{\mathfrak{D}}$ by using the Algorithm 1. Fig. 9 shows the result.

Fig. 9. Conveted EpFSA plant models for the example



As in $G^1_{\mathfrak{E}}$ the value $\bar{x} = 4$ is not valid with respect to the update formula in the transition $q_1 \xrightarrow{b:(x'=x+1)} q_0$, because it would assign $\bar{x}' = 5$ to x and it is not in $Dom(x)$, then this combination is not added to $G^1_{\mathfrak{D}}$, while all others ($\bar{x} = -1, \ldots, \bar{x} = 3$) are added. The same occurs with the transition $q_2 \xrightarrow{c:(x'=x-1)} q_3$, that prevents c_{x-1} to be added to $G^2_{\mathfrak{D}}$.

2) Context Switching: Note that the models resulting from Algorithm 1 require to be complemented with an context switcher (filter model), for them to behave properly. Without the filter, events of an EpFSA may become ambiguous with respect to the source event they represent in a DES plant. For example, in Fig. 5 the occurrence of events created for b and c in Δ, i.e., events in Δ^b and Δ^c, is ambiguous with respect to the occurrence of the source events b and c in Σ.

To avoid such a problem, Algorithm 2 uses the updates of the plant model $G_{\mathfrak{E}}$ to create an additional EpFSA $H_{\mathfrak{D}}$, named filter, that implements context switching based on how variables in V change their values upon transitions. As calculated

Algorithm 1: CONTEXT EXTRACTION FROM SPFSA TO EPFSA

input : $G^i_{\mathfrak{E}} = (\Sigma, V, Q, Q^\circ, Q^\omega, P_V, \gamma)$
output: $G^i_{\mathfrak{D}} = \left(\Delta_{\mathfrak{D}}, Q_{\mathfrak{D}}, q^\circ_{\mathfrak{D}}, Q^\omega_{\mathfrak{D}}, \gamma_{\mathfrak{D}}\right)$

1 **begin**
2 $Q_{\mathfrak{D}} \leftarrow Q$, $q^\circ_{\mathfrak{D}} \leftarrow Q^\circ$, $Q^\omega_{\mathfrak{D}} \leftarrow Q^\omega$,$\Delta_{\mathfrak{D}} \leftarrow \emptyset$,$\gamma_{\mathfrak{D}} \leftarrow \emptyset$
3 **for** *each event* $\sigma_i \in \Sigma$ **do**
4 $\Delta_{\sigma_i} \leftarrow \emptyset$
5 **end**
6 **for** *each active transition* $q_1 \xrightarrow{\sigma:p} q_2 \in \gamma$, *such that* $q_1, q_2 \in Q$, $\sigma \in \Sigma$, *and* $p \in P_V$ **do**
7 **for** *each valuation* $\hat{v} \in Dom(V)$ *that is valid with respect to update formula* p, *such that* $\hat{v}' = p(\hat{v})$ **do**
8 **for** *each value* $\bar{v} \in \hat{v}$, *such that* $\bar{v} \neq \bar{v}'$ **do**
9 $\Delta^\sigma \leftarrow \Delta^\sigma \cup \{\sigma_{v\bar{v}}\}$
10 $\gamma_{\mathfrak{D}} \leftarrow \gamma_{\mathfrak{D}} \cup \left\{q_1 \xrightarrow{\sigma_{v\bar{v}}} q_2\right\}$
11 **end**
12 **end**
13 **end**
14 **for** *each passive transition* $q_1 \xrightarrow{\sigma} q_2 \in \gamma$, *such that* $q_1, q_2 \in Q$, $\sigma \in \Sigma$ **do**
15 **if** $\Delta^\sigma = \emptyset$ **then**
16 $\Delta^\sigma \leftarrow \Delta^\sigma \cup \{\sigma\}$
17 $\gamma_{\mathfrak{D}} \leftarrow \gamma_{\mathfrak{D}} \cup \left\{q_1 \xrightarrow{\sigma} q_2\right\}$
18 **end**
19 **else**
20 **for** *each parameterized event* $\sigma_{v\bar{v}} \in \Delta^\sigma$ **do**
21 $\gamma_{\mathfrak{D}} \leftarrow \gamma_{\mathfrak{D}} \cup \left\{q_1 \xrightarrow{\sigma_{v\bar{v}}} q_2\right\}$
22 **end**
23 **end**
24 **end**
25 **for** *each event* $\sigma_i \in \Sigma$ **do**
26 $\Delta_{\mathfrak{D}} \leftarrow \Delta_{\mathfrak{D}} \cup \{\Delta_{\sigma_i}\}$
27 **end**
28 **return** $G^i_{\mathfrak{D}}$
29 **end**

by Algorithm 2, $H_{\mathfrak{D}}$ can be exposed as the composition of several modular filters H_{v_i}, each one addressing a particular variable $v_i \in V$. Algorithm 2 describes the construction of each EpFSA $H_{v_i} = \left(\Delta_{v_i}, Q_{v_i}, q^\circ_{v_i}, Q^\omega_{v_i}, \gamma_{v_i}\right)$ and it is explained as follows.

For each variable $v_i \in V$ it is created a state $q_{\overline{v_i}}$, denoting every corresponding value possibly assumed by v_i. The initial state is set as $q_{v^o_i}$ (line 5). Then, for every active transition of the input SpFSA $G_{\mathfrak{E}}$, and for each valuation $\hat{v} \in Dom(V)$ that is valid with respect to its update formulas, a corresponding transition is created in the EpFSA H_{v_i} (line 11) and it is labeled with a new parameterized event $\sigma_{v_i\overline{v_i}}$ (line 10). Updates with equal $\hat{v}$ and $\hat{v}'$ do not lead to a new transition, as they just keep context without any change.

By repeating this procedure for all transitions, variables and possible variable values, one obtains a set of EpFSA $H = \{H_{v_1}, \ldots, H_{v_n}\}$, that can be composed to form $H_{\mathfrak{D}} = H^{\|}$. When composed to the plant model $G_{\mathfrak{D}}$, $H_{\mathfrak{D}}$ represents equivalently the context updated by variables and formulas in $G_{\mathfrak{E}}$, using a different construction.

Proposition 1: Let $G_{\mathfrak{D}} \| H_{\mathfrak{D}}$ be calculated as in algorithms 1 and 2, for an input plant $G_{\mathfrak{E}}$ that models G with variables. Then, $\mathcal{L}(\Pi(G_{\mathfrak{D}} \| H_{\mathfrak{D}})) = L(G_{\mathfrak{E}^\bullet})$.

Proof: This trivially follows from (1), (2) and (3). □

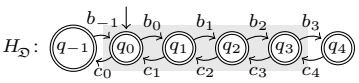
Algorithm 2: SWITCHING EXTRACTION FROM SPFSA TO EPFSA

input : $G_{\mathfrak{E}} = (\Sigma, V, Q, Q^{\circ}, Q^{\omega}, P_V, \gamma)$
output: $H_{\mathfrak{D}}$

1 **begin**
2 $H \leftarrow \emptyset$
3 **for** *each variable* $v_i \in V$, *such that* $Dom(v_i) = \{\overline{v_i}_1, \cdots, \overline{v_i}_n\}$ **do**
4 $Q_{v_i} \leftarrow \{q_{\overline{v_i}_1}, \cdots, q_{\overline{v_i}_n}\}$
5 $q^{\circ}_{v_i} \leftarrow q_{v_i^o}$, $Q^{\omega}_{v_i} \leftarrow Q_{v_i}$
6 $\Delta_{v_i} \leftarrow \emptyset$,$\gamma_{v_i} \leftarrow \emptyset$
7 **for** *each transition* $q_1 \xrightarrow{\sigma:p} q_2 \in \gamma$, *such that* $q_1, q_2 \in Q$, $\sigma \in \Sigma$, *and* $p \in P_V$ **do**
8 **for** *each valuation* $\hat{v} \in Dom(V)$ *that is valid with respect to update formula* p, *such that* $\hat{v}' = p(\hat{v})$ **do**
9 **for** *each value* $\overline{v_i} \in \hat{v}$, *such that* $\overline{v_i} \neq \overline{v_i}'$ **do**
10 $\Delta_{v_i} \leftarrow \Delta_{v_i} \cup \{\sigma_{v_i \overline{v_i}}\}$
11 $\gamma_{v_i} \leftarrow \gamma_{v_i} \cup \left\{ q_{\overline{v_i}} \xrightarrow{\sigma_{v_i \overline{v_i}}} q_{\overline{v_i}'} \right\}$
12 **end**
13 **end**
14 **end**
15 $H_{v_i} = (\Delta_{v_i}, Q_{v_i}, q^{\circ}_{v_i}, Q^{\omega}_{v_i}, \gamma_{v_i})$
16 $H = H \cup \{H_{v_i}\}$
17 **end**
18 $H_{\mathfrak{D}} = H^{\|}$
19 **return** $H_{\mathfrak{D}}$
20 **end**

a) Example: For the inputs $G^1_{\mathfrak{E}}$ and $G^2_{\mathfrak{E}}$ (Fig. 3), we obtain the corresponding EpFSA $H_{\mathfrak{D}} = H_x$, as $V = \{x\}$, by using the Algorithm 2. Fig. 10 shows the result.

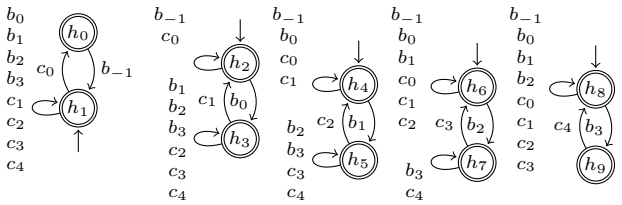
Fig. 10. Filter $H_{\mathfrak{D}}$



Remark that each value in $Dom(x) = \{-1, \ldots, 4\}$ leads to a state in $H_{\mathfrak{D}}$. The values -1 and 4, and equivalently the states q_{-1} and q_4 (highlight in Fig. 10), represent the possibility of underflow and overflow in the plant, so that they are expected to become unreachable when specifications are composed.

As $H_{\mathfrak{D}}$ in Fig. 10 includes an appropriate non-homonym alphabet, it can be modularized as in Fig. 11.

Fig. 11. Modular $H_{\mathfrak{D}}$



The modular version of $H_{\mathfrak{D}}$ can be quite advantageous for abstraction-based synthesis [10], modular synthesis [13], and implementation strategies [12].

B. Control extraction

One remains to show how control rules can be extracted from SpFSA-defined models to control-equivalent EpFSA. This construction is structured by the Algorithm 3. For each model ${E_{\mathfrak{E}}}^i$, it is created a corresponding event-parameterized model ${E_{\mathfrak{D}}}^i$ that expresses the same control rules. However, instead of disabling transitions using test formulas, ${E_{\mathfrak{D}}}^i$ exploits the dilated alphabet to reproduce the same effect.

To construct ${E_{\mathfrak{D}}}^i$, the Algorithm 3 applies a very simple strategy: it reads ${E_{\mathfrak{E}}}^i$ and identifies the variable value that have been prohibited by test formulas. Then, it disables in ${E_{\mathfrak{D}}}^i$ transitions including events that correspond exactly to those variable values.

Algorithm 3: CONTROL EXTRACTION FROM SPFSA TO EPFSA

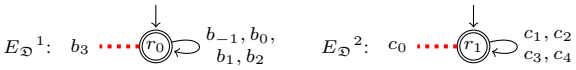
input : ${E_{\mathfrak{E}}}^i = (\Sigma, V, Q, Q^{\circ}, Q^{\omega}, P_V, \gamma)$
output: ${E_{\mathfrak{D}}}^i = (\Delta_e, Q_e, q^{\circ}_e, Q^{\omega}_e, \gamma_e)$

1 **begin**
2 $Q_e \leftarrow Q$, $q^{\circ}_e \leftarrow Q^{\circ}$, $Q^{\omega}_e \leftarrow Q^{\omega}$,$\Delta_e \leftarrow \emptyset$,$\gamma_e \leftarrow \emptyset$
3 **for** *each transition* $q_1 \xrightarrow{\sigma:p} q_2 \in \gamma$, *such that* $q_1, q_2 \in Q$, $\sigma \in \Sigma$, *and* $p \in P_V$ **do**
4 **for** *each valuation* $\hat{v} \in Dom(V)$ **do**
5 **for** *each value* $\bar{v} \in \hat{v}$ **do**
6 $\Delta_e \leftarrow \Delta_e \cup \{\sigma_{v\bar{v}}\}$
7 **end**
8 **end**
9 **for** *each valuation* $\hat{v} \in Dom(V)$, *such that* $p(\hat{v}) = true$ **do**
10 **for** *each value* $\bar{v} \in \hat{v}$ **do**
11 $\gamma_e \leftarrow \gamma_e \cup \left\{ q_1 \xrightarrow{\sigma_{v\bar{v}}} q_2 \right\}$
12 **end**
13 **end**
14 **end**
15 **return** ${E_{\mathfrak{D}}}^i$
16 **end**

Initially, Algorithm 3 creates a structure of states identical to the input SpFSA and starts the alphabet and transition relation of ${E_{\mathfrak{D}}}^i$ as empty. Next, it reads all transitions from ${E_{\mathfrak{E}}}^i$ to create the alphabet and transition relation for the output ${E_{\mathfrak{D}}}^i$. For each value $\bar{v}$, of each valuation $\hat{v} \in Dom(V)$, a corresponding event $\sigma_{v\bar{v}}$ (line 6) is created and added to event set. Then, a transition is created for each events that is enabled by the specification (test formula is true). The others are disabled. Each valuation $\hat{v} \in Dom(V)$ that makes a test formula *true*, is associated with a transition in ${E_{\mathfrak{D}}}^i$ by labeling it with the corresponding event $\sigma_{v\bar{v}}$ (line 11).

a) Example: The result of converting ${E_{\mathfrak{E}}}^1$ and ${E_{\mathfrak{E}}}^2$ from Fig. 3 into the EpFSA ${E_{\mathfrak{D}}}^1$ and ${E_{\mathfrak{D}}}^2$, is shown in Fig. 12.

Fig. 12. Converted EpFSA specification models for the example



Note that, in $E_{\mathfrak{E}}$, the formula $x < 3$ is false for $\bar{x} = 3$ and $\bar{x} = 4$. Similarly, the formula $x > 0$ is false for $\bar{x} = -1$ and 0. Therefore, the events that correspond to these combinations are disabled by the converted model $E_{\mathfrak{D}}$, so that they impose the same control rule to the plant.

C. Analysis of results

In this section, we provide a numerical analysis that gives a hint about the benefits that our conversion method aggregate to the practice of control engineering of DES.

Table I shows the number of states unfolded by the models that represent plant, specification and synthesis input, for each modeling approach exploited in this paper. We assume number of states as a fair interpretation of the different spheres of complexity in DES, as it in general gives an idea about the effort to obtain and process models.

The buffering example is again used to quantify the three modeling approaches, for which it is assumed that plant models aim to express the same behavior and specification models intend to impose equivalent control rule over the plant.

TABLE I
COMPARISON AMONG MODELING APPROACHES APPLIED TO THE EXAMPLE

Model	**FSA**		**SpFSA**		**EpFSA**	
Plant	G:	4	$G_{\mathfrak{E}}$:	4	$G_{\mathfrak{D}} \| H_{\mathfrak{D}}$:	24
Specification	E:	4	$E_{\mathfrak{D}}$:	1	$E_{\mathfrak{D}}$:	1
Synth. Input	K:	16	$K_{\mathfrak{E}^\bullet}$:	16	$K_{\mathfrak{D}}$:	16

Note, in the last row, that all approaches lead to a synthesis input with the same number of states. By Prop. 1, it follows that $\mathcal{L}(K) = \mathcal{L}(K_{\mathfrak{E}^\bullet}) = \mathcal{L}(\Pi(K_{\mathfrak{D}}))$. In fact, their differences rely only on the way modeling is conducted.

In the column FSA, K results from G and E which are totally non-automated, i.e., their modeling depends entirely on the engineer. As a result, E (second row) has more states (4 states) than the other parameterized versions (1 state).

Differently, SpFSA reduce the model E from 4 to 1 state in $E_{\mathfrak{E}}$, at the price of designing update formulas in the plant, which in general is a simple task. Similarly, $E_{\mathfrak{D}}$ (column EpFSA) has also a single state, at the price of designing the filter and compose it to the plant. In both cases, $E_{\mathfrak{E}}$ and $E_{\mathfrak{D}}$ remain with 1 state when the buffer size n changes, while E has always $n+1$ states.

The problem involving SpFSA and EpFSA is that, individually, they do not provide a complete mechanism to both simplify modeling and lead to efficient modular synthesis inputs. On one hand, SpFSA are modeling friendly, but they are not directly modular. On the other hand, EpFSA are more suitable for modularization, but they require to implement manually the update semantic.

Therefore, the conversion method proposed in this paper combines the advantage of designing $G_{\mathfrak{E}}$ using only 4 states, while providing an event-parameterized output that is suitable for the modular abstraction-based synthesis framework in [13].

IV. CONCLUSION

This paper discussed how parameters can simplify modeling and control of DESs. Two methods are presented, each one constructed on a specific framework, so that they do not directly combine advantages. A conversion method is then proposed to allow conducting modeling using a design-friendly approach, and systematically migrate from this framework to another, more suitable for synthesis.

The algorithms presented in this paper are all polynomials in the number of states of the input automata models. An example of a buffering system illustrates the approach.

Prospects of future research aim to assess possible advantages of our conversion method over implementation issues. We also intend to provide tooling support and test the approach on more complex examples.

ACKNOWLEDGES

This work was supported by CNPq, under grant number 402145/2016-0, 09, Araucaria Foundation, CAPES, and FINEP, and partially supported by ERDF - The European Regional Development Fund through the Operational Programme for Competitiveness and Internationalisation - COMPETE 2020 Programme, and by National Funds through FCT - Fundação para a Ciência e a Tecnologia, within project POCI-01-0145-FEDER-030947 (KLEE).

REFERENCES

[1] B. Esmaeilian, S. Behdad, and B. Wang, "The evolution and future of manufacturing: A review," *Journal of Manufacturing Systems*, vol. 39, pp. 79 – 100, 2016.

[2] A. L. Silva, R. Ribeiro, and M. Teixeira, "Modeling and control of flexible context-dependent manufacturing systems," *Information Sciences*, vol. 421, pp. 1 – 14, 2017.

[3] Y.-J. Chen, K.-S. Hwang, and W.-C. Jiang, "Policy sharing between multiple mobile robots using decision trees," *Information Sciences*, vol. 234, pp. 112 – 120, 2013.

[4] Y. Qamsane, M. El Hamlaoui, T. Abdelouahed, and A. Philippot, "A model-based transformation method to design plc-based control of discrete automated manufacturing systems."

[5] C. G. Cassandras and S. Lafortune, *Introduction to discrete event systems*. Springer Science & Business Media, 2009.

[6] P. J. Ramadge and W. M. Wonham, "The control of discrete event systems," *Proceedings of the IEEE*, vol. 77, no. 1, pp. 81–98, 1989.

[7] R. Malik and M. Teixeira, "Modular supervisor synthesis for extended finite-state machines subject to controllability," in *Discrete Event Systems (WODES), 2016 13th International Workshop on*. IEEE, 2016, pp. 91–96.

[8] S. Mohajerani, R. Malik, and M. Fabian, "Compositional synthesis of supervisors in the form of state machines and state maps," *Automatica*, vol. 76, pp. 277 – 281, 2017.

[9] M. Teixeira, R. Malik, J. E. Cury, and M. H. de Queiroz, "Supervisory control of des with extended finite-state machines and variable abstraction," *IEEE Transactions on Automatic Control*, vol. 60, no. 1, pp. 118–129, 2015.

[10] J. E. Cury, M. H. de Queiroz, G. Bouzon, and M. Teixeira, "Supervisory control of discrete event systems with distinguishers," *Automatica*, vol. 56, pp. 93–104, 2015.

[11] Y.-L. Chen and F. Lin, "Safety control of discrete event systems using finite state machines with parameters," in *American Control Conference, 2001. Proceedings of the 2001*, vol. 2. IEEE, 2001, pp. 975–980.

[12] M. Rosa, M. Teixeira, G. W. Denardin, C. R. C. Torrico, and J. E. R. Cury, "Efficient implementation of distinguished controllers for discrete-event systems," in *IFAC World Congress, WC'17*, Toulouse, France, 2017, pp. 1215–1220.

[13] M. Teixeira, J. E. R. Cury, and M. H. de Queiroz, "Exploiting distinguishers in local modular control of discrete-event systems," *IEEE Transactions on Automation Science and Engineering*, vol. 15, no. 3, pp. 1431–1437, July 2018.

[14] M. Rosa, M. Teixeira, and R. Malik, "Exploiting approximations in supervisory control with distinguishers," in *International Workshop on Discrete Event Systems*, Sorrento, Italy, 2018.

[15] P. Gohari and W. M. Wonham, "On the complexity of supervisory control design in the RW framework," *IEEE Transactions on Systems, Man, and Cybernetics, Part B*, vol. 30, no. 5, pp. 643–652, 2000.